RN-Wissen.de - Bascom Libraries

from RN-knowledge, the free knowledge base

Contents

Bascom Libraries

In the installation area of Bascom has a series libraries with the endings ".LBX" and ".LIB". The former are the same, but pre-compiled versions of the ".lib" files. This precompile happens in the LIB Manager (Menu-> Tools).

But the ".LBX" Files are no Object libraries, as we know from other languages ago and are joined by a "linker". Both are plain text files, which are only tatächlich translated when compiling the Basic main program in machine code.

Therefore, it is Bascom not really care whether at $ lib statement "lib.lib" or "lib.lbx" indicating it is basically the same work

Why, then, in general, ".LBX"?

By precompiling the libraries are still somewhat illegible, and if one of the Library ".LIB" version is not there, you will probably not give the code.

Both versions are collections of "building blocks" that integrates Bascom only in the user program when they are actually needed. These blocks look like this:

```
[BAUSTEIN_NAME]
BAUSTEIN_NAME:
.... Assembler Anweisungen ...
RET
[End]
```

Example

Man with Notepad or another editor creates a file "Testlib.LIB" in the library directory of Bascom (must check in the installation directory where that is)

```
[My_function]
My_function:
  RET
[End]
```

(And save, of course)

In the main program (in Bascom) to write now

```
$ Regfile = "m32def.dat"
$ Crystal = .....
$ Lib "testlib.lib"

$ External my_function
Declare Sub my_function ()

'.... Somewhere ...
CALL my_function
```

As a result, the function is added to the next free location in the machine code and called on "Call".

It now needs only anything meaningful in the reinzuschreiben function.

Replacing Bascom Library Functions

It is very easy, functions that are included in any Bascom library to replace with your own code. Bascom goes namely when searching for a function currently required in the order of "$ LIB" statement before. The Standard Library "MCS.LBX" is always searched first last. If you write so

```
 $ LIB "testlib.lib"
```

he seeks only to that library, and only when there is nothing drinsteht, he takes what is in MCS.LIB.

Unfortunately, all Bascom functions in libraries are not present, much is permanently installed in Bascom program. And some can be found only after a little detective work ("PULSEIN" means, for example "_PULSE_IN").

Tips

Approach

If you want to develop a library function, it is convenient to write it first as a normal subroutine with inline assembler, and only after the halfway works to transfer them to a library. The syntax is a little different, but it gets you out. But the advantage is that you can debug its assembler code in the simulator normal, and that's worth something.

Arguments

If you want to pass the function arguments or a resultant address, one must visit the Bascom Call-standard note. But he is quite complicated. All (global) variables that you have created with DIM, you can also address with "LOADADR", saving stuff.

Secure

Between "ASM $" and "$ END ASM" man is ruler over the entire controller. You can use all the stops uncontrollably. So it's not like in an ISR that you have to backup everything and restore

Only the register

```
 YL: YH
R4: R5
R6
```

you should back up and restore at the end, if you want to change it.

Tutorial: Creating a library function

It should be taken once the whole process of development.

Mem_move

This example function is simply a SRAM area to copy to another, the goal is always a byte array, however, the source should be data with arbitrary data types. The transfer length must be therefore of course specify. As a result, in turn, the number of bytes transferred. This is of course meaningless in this example, but we want to see how you can return a value, too.

```
Declare Function Mem_move (target As Byte, srcaddr As Word, byval Length As Byte) As Byte
```

So far everything clear, "target" is the target array, "Length", the transmission length, but what's the deal with *"srcaddr As Word"* is all about? There are supposed to be transferred any data, not just a WORD.

That is a hook at Bascom not going quite as directly.

We need the function declaration so filled with a specific data type, and type "something" there's just not.

So you do something with "the address of the address" (C programmers know that). That is, we define a WORD, write purely the current data source address, and our function always gets the address of this Word.

```
Dim Source As Word

   Source = VarPtr (source_string) 'The SRAM address of "Source_String"
```

Example main program

(For a Atmeg32, but that does not matter). A few things are still in the process, so you can control the situation also in the simulator terminal with "print".

```
$ Regfile = "m32def.dat"
$ Crystal = 8000000

$ Baud = 9600
$ Hwstack = 64
$ Swstack = 256
$ Frame size = 64


Declare Function Mem_move (target As Byte, srcaddr As Word, byval Length As Byte) As Byte
```

```
Dim Ziel_len As Byte 'as to the result (length) pure
Dim Ziel_arr (24) As Byte 'here should be copied.

Dim Print_string As String * 24 At Ziel_arr overlay 'that's just for the "PRINT" to the
                                                    Herzuzeigen 'target array on simple

Dim Source_string As String * 24 'which are the target data

   Source_string = "Hello, world" 'thus what it says

Dim Source As Word 'see above

   Source = VarPtr (source_string)

   Ziel_len = Mem_move (ziel_arr (1) Source: 5) 'of the function call:
                                                    Copy '5 bytes of source

   Print "From";  Source_string 'so does the source string
   Print "To (";Str (ziel_len); ")";  Print_string 'and so the copied data

End 'no loop, including?
```

Step 1: Inline assembler

The function arguments:

Again, the reference to the Bascom call standard . At the beginning of this function takes its arguments in the "soft stack", the pointer register pair Y (YL: YH) points to the beginning.

Bascom specifies the addresses of the arguments in the following descending order on the soft stack:

```
Function Mem_move (target As Byte, srcaddr As Word, byval Length As Byte) As Byte
```

Only the address of the function result, then starting from the left, the argument addresses in the bracket, making a total of:

- High (Result)
- Low (Result)
- High (Target)
- Low (Target)
- High (srcaddr)
- Low (srcaddr)
- High (Length)
- Low (Length)

And on this last value indicates YL: YH when calling the function. Read is that with

```
LD register, Y + offset
```

An example:

```
   ldi r24, 65
   ldd xl, y + 6
   ldd xh, y + 7
   st x, r24
```

Corresponds to the Bascom statement

```
Mem_move = 65
```

ie, the function would return the result 65

And now the "inline"

```
Function Mem_move (target As Byte, srcaddr As Word, byval Length As Byte) As Byte
   $ Asm
   ldd xl, y + 0 'length addr lo
   ldd xh, y + 1 'length addr high
   ld r24, x 'Length => R24
   clr r25 'Delete count register
  ! And r24, r24 'length = 0?
   BREQ Mem_mov_xit '= 0 when the length is to do garnix
```

First we Xl: XH loaded with the address of the length of the argument, then register 24 with the actual value. The counter for the function result is set to zero, then the specified length is checked if at all what to do.

(Note: the call sign when "and" shows the Bascom that this is an "assembler And" and concerns him nothing)

```
   ldd xl, y + 2 'source PARAM
   ldd xh, y + 3 'source PARAM
   ld zl, x + 'source
   ld zh, x 'source
```

"Source" is more a corner: on the stack is the address of a befindete WORD. And there it was only then is the actual address of the source data.

```
   ldd xl, y + 4 'target
   ldd xh, y + 5 'target
```

This is the "normal case". On the stack is directly the destination address (Ziel_arr (1))

```
Mem_mov_lop:
   ld r22, Z + 'read source
   st x + r22 'write target
   inc r25 'counter ++
   dec r24 'length--
   brne Mem_mov_lop '<> 0?  => Loop
```

The Copy-loop: Z to auto increment the register 22, thence to X, with increment. Every time we increased the result counter and reduces the control counter.

```
Mem_mov_xit:
   ldd xl, y + 6 'result
   ldd xh, y + 7 'result
   st x, r25
   $ End Asm
End Function
```

X is loaded with the address of the function result (see above) and R25 is written down there.

Testing in the simulator you should look at the actual variables addresses from the ".rpt" file. This and the Registrar and memory window of the simulator can be quite well observe the individual steps. For the printout's make a check mark in the "Terminal"!

Step 2: Library Function

Suppose we are satisfied, then we now have a library. I do this with Word or Notepad, a text editor does not matter, stop. Goes well with Bascom itself, but is a matter of taste

An empty Library

```
Anybody copyright =
www = http://www.roboternetz.de
email = picnick@nowhere.at
BASCOM AVR compiler comment = library for demo
libversion = 0.1
date = 04 jun 2006
history = Beta
```

And now inserters function with cut & paste from the Bascom editor

```
Anybody copyright =
www = http://www.roboternetz.de
email = picnick@nowhere.at
BASCOM AVR compiler comment = library for demo
libversion = 0.1
date = 04 jun 2006
history = Beta

;  Declare Function Mem_move (target As Byte, srcaddr As Word, byval Length As Byte) As Byte

[Mem_move]
Mem_move:
   ldd xl, y + 0 'length
   ldd xh, y + 1 'length
   ld r24, x
   clr r25 'clearCounter
   and r24, r24 'length = 0?
   BREQ Mem_mov_xit 'no action

   ldd xl, y + 2 'source PARAM
   ldd xh, y + 3 'source PARAM
   ld zl, x + 'source
   ld zh, x 'source

   ldd xl, y + 4 'target
   ldd xh, y + 5 'target

Mem_mov_lop:
   ld r22, Z + 'read source
   st x + r22 'write target
   inc r25 'counter ++
   dec r24 'length--
   brne Mem_mov_lop '<> 0?  => Loop

Mem_mov_xit:
   ldd xl, y + 6 'result
   ldd xh, y + 7 'result
   st x, r25

  ret;  ready
[End]
```

The call sign at "AND" we have gone, but comes the added. I advise to write the function declaration as a comment.

```
  [Mem_move]
Mem_move:
.....
 ret;  ready
[End]
```

The "RET" has made in-line version of the Bascom ("END FUNCTION"), which must not be forgotten now.

And now "save as" RN_LIBRARY.LIB at the Bascom other libraries. With me is the example

```
 E: \ Program Files \ MCS Electronics \ BASCOM-AVR \ LIB \ RN_LIBRARY.LIB
Watching with the file type!
```

We can now already check in at Bascom "tool" is a Lib-Manager, which now must use this library to be seen.

The **rebuilt main program** now looks like this

```
$ Regfile = "m32def.dat"
$ Crystal = 8000000

$ Baud = 9600
$ Hwstack = 64
$ Swstack = 256
$ Frame size = 64

$ Lib "RN_Library.LIB" 'this is new
$ External Mem_move 'is new

Declare Function Mem_move (target As Byte, srcaddr As Word, byval Length As Byte) As Byte


Ziel_len As Byte Dim
Dim Ziel_arr (24) As Byte

Dim Print_string As String * 24 At Ziel_arr overlay
Dim Source_string As String * 24

Dim Source As Word

   Source_string = "Hello, world"

   Source = VarPtr (source_string)
   Ziel_len = Mem_move (ziel_arr (1), source, 5)

   Print "From";  Source_string
   Print "To ("; Str (ziel_len); ")";  Print_string

End
```

If we now press "F7", both the Basic program and the library is compiled.

```
 That's actually
```

LBX?

Because of ".LBX" If we make it compile with the Lib Manager Library, the result looks like this:

```
Comment = Compiled LIB file, no comment included

Anybody copyright =
www = http://www.roboternetz.de
email = PicNick@nowhere.at
BASCOM AVR compiler comment = library for demo
libversion = 0.1
date = 04 jun 2006
history = Beta
[Mem_move]
Mem_move:
.CBJ 81A8
.CBJ 81B9
.CBJ 91Bc
```

```
.OBJ 2799
.OBJ 2388
   BREQ Mem_mov_xit 'no action
.OBJ 81AA
.OBJ 81BB
.OBJ 91ED
.OBJ 91FC
.OBJ 81AC
.OBJ 81BD
Mem_mov_lop:
.OBJ 9161
.OBJ 936D
.OBJ 9593
.OBJ 958A
   brne Mem_mov_lop '<> 0?  => Loop
Mem_mov_xit:
.OBJ 81AE
.OBJ 81BF
.OBJ 939C
.OBJ 9508
[End]
```

So you can be know-how to hide from prying eyes. To use this, you must be in the main program but then write:

```
$ LIB "RN_LIBRARY.LBX"
```

Note

Library functions can also be used without the call standard. That is the thing with the arguments handover via the soft stack is not mandatory.

You could also do the following and save some code that the building of the soft stacks and reading the arguments in the function itself are quite expensive.

In our example, so the copy on the pointer register Z u ran in the core. X with the counter r24

Main program

```
$ LIB = "Library.lib"
$ External move_z_to_x
Declare Sub move_z_to_x ()

     $ Asm
     LDI r24, 5
     $ End asm
     LOADADR array (1), X 'are the only two machine instructions!
     LOADADR source_string, Z 'as well
     GOSUB move_z_to_x 'is a
```

Library.lib

```
...
[Move_z_to_x]
move_z_to_x:
_mov_lop:
   ld r22, Z + 'read source
   st x + r22 'write target
   dec r24 'length==
   brne _mov_lop '<> 0?  => Loop
   ret
[End]
```

Author

- Picnick

See also

- Bascom
- Bascom Inside

From " http://rn-wissen.de/wiki/index.php?title=Bascom_Libraries&oldid=14480 "

Categories : Microcontrollers | Basics | Software | Practice | Source Bascom